

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
7 September 2001 (07.09.2001)

PCT

(10) International Publication Number
WO 01/65366 A1

(51) International Patent Classification⁷: G06F 9/45,
11/30, 12/14, H04L 9/00, 9/32, H04K 1/00

(74) Agents: ROSENTHAL, Lawrence et al.; Stroock &
Stroock & Stroock & Lavan LLP, 180 Maiden Lane, New
York, NY 10038 (US).

(21) International Application Number: PCT/US01/06599

(22) International Filing Date: 1 March 2001 (01.03.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/186,538 2 March 2000 (02.03.2000) US

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU,
AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ,
DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR,
HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR,
LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ,
NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM,
TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (*regional*): ARIPO patent (GH, GM,
KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian
patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European
patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,
IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF,
CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

(72) Inventors; and

(75) Inventors/Applicants (*for US only*): METLITSKI,
Evgueny, A. [RU/US]; 56 Des Moines Court, Tinton
Falls, NJ 07712 (US). UTYANSKI, Dmitry, B. [RU/RU];
Apartment (KZ)57, 88 Morisa Toreza Avenue, St. Peters-
burg (RU).

Published:

— with international search report

*For two-letter codes and other abbreviations, refer to the "Guid-
ance Notes on Codes and Abbreviations" appearing at the begin-
ning of each regular issue of the PCT Gazette.*

(54) Title: SYSTEM AND METHOD FOR PROCESS PROTECTION

(57) Abstract: A method of developing a protected software application comprises identifying segments of the application to be protected and compiling those segments according to a protected instruction set. The protected segments are then linked together with the unprotected segments for distribution. The protected segments can only be executed using a trusted module connected to a computing device. The trusted module comprises a virtual machine programmed with an instruction set corresponding to the set of instructions used to compile the protected segments. Using the trusted module, the state vector of the process of execution is monitored to insure integrity of the process. Various encryption and technological security measure may also be used.

WO 01/65366 A1

executed. Special systems of applied hardware and software have been used by hackers to tamper with others' programs and data, such as debuggers, logic analyzers and emulators, to name a few. Although these devices were developed for programmers to aid in the development and debugging of their applications, they have proved to be allies to hackers and thieves.

While many solutions to protect programs and data have been developed, as set forth in further detail below, the potential for influences on processes during the execution of applications remains a primary concern regarding the security of computer platforms. Openness of computer platforms and potential for modification of processes remain the most troublesome of the problems in information security. For instance, by influencing a process, a hacker can influence or modify the results or output of an application, thereby potentially costing the users valuable time and money.

The main objectives of software protection systems include:

- a) impossibility to reveal the algorithm of a program;
- b) impossibility of any unauthorized influence on the process of program execution, which changes the logic of the program and/or protocol of program interaction with user;
- c) guaranteed correctness of time parameters, related to user actions, which are determined by the program logic;
- d) impossibility of arbitrary changes of the key data or parameters of the program and/or results of its work;
- e) authenticity of the process and the process name.

Integral in these objectives is the term "process", which is fundamental to Computer Science. While a program itself is a static set of directives, execution of the program is a dynamic activity whose features change in time. This activity is called a "process". The state of a process at any given time is generally referred to as the "process state". The difference

by a state machine. While this "dynamic encryption" improves system security against "record and playback" attack, it does not fully address process protection.

Cryptographic protection systems using a unique key built into the microprocessor are described in U.S. Patent Nos. 5,034,980 to Kubota and 4,633,388 to Chiu. In such systems, the microprocessor executes a program whose code has been enciphered beforehand using the indicated unique key, i.e. the processor can run only the programs which have been prepared specially for it. While such hardware security means provide a high level of protection, they depend upon expensive IC manufacturing technologies and are not flexible and universal enough for a widespread public use.

In systems where the protection means are built into such functional components of the computer, such as the processor, memory, or input/output controllers, it is necessary to provide physical integrity of those components. Such a solution is described in U.S. Patent No. 5,007,083 to Constant. Although this solution can be used in "special-purpose" systems, such as government or military systems developed for a particular, specialized use, it is of little use in open architecture computers, such as personal computers.

Protection systems combining hardware and software means have been the most widely adopted, since they are the most universal and available for public use. Protection of software against unauthorized copying or use can be provided, for example, using direct protection of one of the memory devices of computer containing protected programs and data. Such a solution is described in U.S. Patent Nos. 5,081,675 to Kittirutsunetorn and 5,533,125 to Bensimon et al. These protection systems use special devices (i.e., coprocessors, electronic keys, cartridges) connected to one of the ports of computer. In simple systems, such devices store codes of keys (e.g., electronic keys) used for authentication of a copy of a software product or enciphering/deciphering of program and data segments. In more advanced systems, secure coprocessors are used to provide secure execution of program code segments.

directly when installing the program on a user computer. As is described below, clock or counter initialization approaches can encounter difficulties related to uncontrollable user reaction time.

A more sophisticated software usage metering system is described in U.S. Patent No. 5,083,309 to Beysson. Such systems make it possible for the software to be rented, with the rental user paying a fee based on the time the software actually ran. The protection mechanism uses a memory card and a card reader associated with the computer to run the software. Various intermediate results are stored in the memory of the card and not in the memory of the computer. Similar to the Shavit system, a substantial disadvantage of this protection method is that the programs' algorithms are open to analysis and possible modification by a malicious user. Enciphering of data transmitted to and from the memory card, as well as ensuring that the time required for the execution of the various portions of the software does not become too long, is insufficient to resist the unauthorized actions. Moreover, ensuring that the time for execution is short is not acceptable for the majority of programs, since they interact with an end user, which introduces unpredictable delays into operation. Disabling of the clock while executing procedures involving interaction with the user, which is described in the Beysson patent to mitigate this problem, compromises the whole execution time metering system.

Another application of cryptography is to provide means to check software code or data authenticity. Usually, this is done using some variation of private/public encryption to enable the loader to verify that the software is indeed provided from the certified source. An example of such an approach is described in U.S. Patent No. 5,724,425 to Chang et al., where software is distributed in a signed "passport", including the software writer's name and license. Only when the relevant information in the "passport" is valid, can the software be executed.

limit the functionality of the on-card program.

As can be seen from the above review, the encryption of data and application code is known. However, to be used on a PC, this encrypted data or code must be transformed into an open form (i.e., it must be decrypted usually on the same PC). As a result, the decryption algorithm, crypto-keys, and all other parameters of deciphering are open and subject to analysis by unauthorized individuals. Smart cards and other hardware solutions have similarly failed to provide adequate protection due to their limited processing power or specialized nature. Thus, while current systems provide a modicum of protection to software code and data, no present system provides comprehensive process protection that is available for widespread public use.

SUMMARY OF THE INVENTION

The present invention overcomes the shortcomings of the prior art. The present invention generally comprises a system and method for providing protection to the processes executed on a computer. Unlike the prior art, the exemplary embodiments and equivalents disclosed herein provides a low cost trusted computer platform that comprises a trusted module connectable to a host computer, such as a personal computer (PC), a personal digital assistant (PDA), or other computing device, that enables the secure execution of an application. The trusted module includes a virtual machine and security kernel upon which all of the protection mechanisms are built. The system is flexible due to the smaller size of the security kernel, which allows for smaller amounts of resources to be available to the kernel. Moreover, because only portions of an application are executed on the trusted module limited processing resources are necessary. It further eliminates any possibility of unauthorized external influence on the processes and supports a wide variety of public applications. Yet further, the present invention provides traditional protection features, such as protection of programs and data from copying or unauthorized access and use. In general,

In the invention these goals are achieved using a special technology of software preparation on its development stage and a special technology of software execution on its operation stage. The technology, on its execution stage involves interaction of a standard computer (for example, IBM PC-compatible) and of the special additional device – Trusted Module, connected to the computer using one of the standard interfaces and including at least the following units: CryptoInterpreter (CI), Reference Clock (RC) and Non-Volatile Random Access Memory (NVRAM). One possible implementation of CryptoInterpreter is a software implementation which assumes that the trusted module should have at least a CPU, ROM containing control program, and RAM.

The use of the PEP technology includes the following steps among others:

- identification of software fragments that should be protected;
- implementation of the identified fragments using a high or low level programming language, their translation to the executable code using special software (CryptoCompiler or CryptoAssembler) and, possibly, additional information such as a key (CryptoKey);
- when necessary -- initialization of non-volatile memory (NVRAM) of the trusted module and writing of the corresponding key information (CryptoKey) into the NVRAM;
- loading and execution of the program on a user computer, when the protected fragments of the program are initially loaded into the memory of the computer as a part of software and then are executed during an interaction of trusted module and computer in one of the modes of work described in this invention, namely:

(a) by a device retrieving and interpreting the executable code and data of the protected fragments in a word-by-word mode as the program is being executed with possible deciphering of commands and data using a deciphering key stored in NVRAM or ROM of the device. Thus, the working data (components of the state vector) of the protected process can be read by the trusted module from computer RAM and transferred by the trusted module

Unlike smart cards, the trusted module of the present invention can act as a key to prevent access to data on another device, securely store data that is accessible to the user and inaccessible to the user, prevent execution of a process, provide identification, authorization, or authentication of the trusted module holder, modify itself, protect itself, and initiate processes based on dynamic instructions.

While the main purpose of a smart card is the predetermined execution of entire processes utilizing known content, the trusted module of the present invention is a closed virtual machine with a dynamic architecture. In other words, the trusted module can process any application, including real-time processes. It can execute internal processes, and at the same time, interact with external machines such as PC's. Furthermore, the execution of joint segments of processes with a PC is possible. The trusted module controls those segments of the process that are executed on an external machine such as the PC. Thus, the processes executed on a PC and interacting with the trusted module also become closed processes.

Unlike smart cards, the trusted module of the present invention has a much greater computing and memory resources, and its internal structure supports the dynamic architecture of the trusted module and other processes and parameters mentioned above.

The present invention can be used in a wide variety of mainstream applications. While the aggressive growth of Business-to-business ("B2B") commerce has created an infrastructure that will enable businesses to save millions of dollars in procurement costs, the new technology has created a vehicle for potential multimillion-dollar fraud and/or theft. The present invention would enable businesses to create a totally secure B2B infrastructure that would eliminate companies' potential exposure and liability. As such, the present invention would enable a secure environment across all components of ERP/XRP.

Furthermore, the present invention would provide a mechanism for the protection of proprietary information of global computing devices. Thus, travelers could confidently bring

type wallet could be created which could not be tampered with because the card would require the physical attachment to an authorized device in order to retrieve any monetary value stored on the card, an authorization process, such as password or biometrics, prior to access to the monetary content, and an inability to remotely access the card.

In distance learning applications, the present invention could provide a secure environment for the administration of "distance tests".

Because the present invention provides protection to the process of an application in its dynamic state in addition to the program code and data, the present invention could provide protection against modification of the infrastructure logic of a PC which allows viruses to transparently travel within PC's. As such, the present invention could provide strong anti-virus protection. The present invention also provides protection against internal hacking and user-identification when digital content is being transferred between users. Yet further, the present invention could be used to create a secure environment for electronic notaries to create an objective record of documents, requisitions, electronic signatures and electronic contracts. Ticketing and on-line postal services offer the possibilities for use of the present invention. Still further, the present invention could be used to create a secure digital information card for identification of the holder. The card could include photographs, facial scans, fingerprints, retinal scan information, general descriptive information, other biometric information or processing capability, and/or passwords. As such, the present invention can be applied to applications such as by way of non-limiting example passports, personal identification, employee ID's, drivers' licenses, credit cards, electronic keys, and access to on-line storage of essential medical, legal or other information.

Other objects and features of the present invention will become apparent from the following detailed description, considered in conjunction with the accompanying system schematics and flow diagrams. It is understood, however, that the drawings, which are not to

Fig. 12 shows the variant of mapping of logical addresses of the address space of the PEP virtual machine to the host computer memory when using the scheme with word-by-word exchange;

Fig. 13 is similar to Fig. 12 and shows in more detail the variant of design of address mapping, using fixed and variable keys as mapping parameters; and

Fig. 14 shows the distribution of the protected process state vector components in the physically protected Trusted Module and the host computer RAM.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

There will now be shown and described in connection with the attached drawing figures exemplary embodiments of a system and method for developing and protecting the processes of a software application.

According to an exemplary embodiment of the present invention, a system comprising a physically secure device in communication with a conventional open architecture computer (e.g., a personal computer) provides a trusted computing platform that protects the processes of an application in its various dynamic states, as well as the programs and data of the application.

In general, with reference to Figures 1-14, a method for developing a protected application 7 for use on an open architecture host computer 1 comprises identifying one or more segments S2, S4, and S6 of the application 7 to be protected and compiling the identified segments using a cryptocompiler into cryptocode. The remaining segments S1, S3, and S5, i.e., those not identified to be protected, are compiled using a conventional compiler 5 into a known form of machine code. The cryptocode and the machine code are then combined using a linker 6 to form the resulting protected application 7. As will be described in further detail below, because the resulting protected application 7 can only be operated by a computer in communication with a secure device capable of executing the cryptocode, the

state is characterized by a multi-component vector SVP $(p_1, p_2, \dots, p_N \mid t_i)$, where $p_n, n \in [1, N]$ is a set of parameters of the process that describes the process state at discrete time points $t_1, \dots, t_i, \dots, T$. The computer's processor, the function of which is to change the process state, performs transfer of the process from one state to the next one. The completed process passes through all its states, from the starting SVP $(p_1, p_2, \dots, p_N \mid t_1)$ to the final SVP $(p_1, p_2, \dots, p_N \mid T)$.

If $N+1$ components of the SVP are considered to be the axes of $(N+1)$ -dimensional space, then the deterministic process is characterized by a unique trajectory in this space.

However, the trajectory of real processes often are not predetermined. A large number of internal and external factors influence the execution of the process resulting in changes to the process's trajectory in the $(N+1)$ -dimensional space of the SVP components. These factors include, but are not limited to, the internal vagueness of processes that use random number generators, interactions with the user, and the timing of events in a real time process. Changes in the trajectory caused by the influence of these and other internal and external factors are distinctive of a real-time process. The set of possible trajectories of the process depicted in Figure 1 are derived from (1) the internal uncertainties of the process (\circ); (2) interactions with the user (Δ); and (3) events which are determined by time (\square). Assuming that the ultimate goal of any process is to provide the user with objective results, the most important events are marked on the graph by the points where the trajectory of the process branches.

Time is one of the factors that define the trajectory of a process. There are two types of time for the computer processes: (1) internal discrete time of the process and (2) external real time of events related to the execution of the process. The relationship between these two types of time is important for the process control. Real time is an objective external factor, which changes independent of the process. Internal time, however, is subject to

6. Control of "disjoints" (when the action on the process is unsanctioned) of the process in areas with predetermined trajectory; termination of the process when deviation from the trajectory is detected ([6] Fig. 2).
7. Protection of predetermined areas of the process in order to protect the elements of the program (models, algorithms, parameters and data structures) ([7] Fig. 2).
8. Protection of the final value of the process state vector to prevent changes to its elements after the process completion ([8] Fig. 2).
9. Control of time intervals between the possible and expected internal and external events (T_1, T_3, T_5, T_6).
10. Control of total execution time of the process T_7 .

With reference to Figure 3, there is shown an illustration of an exemplary embodiment of the system architecture of a trusted module for use with the present invention. The trusted module is a specialized physically protected processing device, which can be communicatively connected to a computer. With further reference to Figures 4 and 5, the trusted module 100 comprises various hardware components and internal firmware designed to interact with a host computer to jointly execute a protected application. The execution process is described in further detail below. Referring again to Figure 3, an exemplary multi-level architecture includes, but is not limited to, the following levels:

- o Hardware platform level (1). At this level there is a hardware implementation of the object machine, which is a microcomputer, as described below.
- o Kernel level (2). At this level there is a Trusted Module Operating System ("TMOS") kernel, which controls usage of internal resources of the trusted module and supports logical security functions of the trusted module. In particular, this level contains all necessary cryptographic functions.

speed of at least 100mhz or higher. The base processor 110 should preferably have an output of over 80 million ins/sec, while the output of the crypto-interpreter 200 (as shown in Figure 5) should have an output of up to about 100K ins/sec. The memory 120 of the trusted module 100 preferably has a volume of memory for programs and data of about 64K words or more. The NVRAM 124 preferably has a memory of about 4 megabytes or more and the memory word is preferably 16-bit. It should be noted, however, that any other bit size such as 16-bit, 24-bit, 32-bit, 64-bit, or 128-bit memory could be utilized.

Furthermore, as will become evident from the following discussion of the process protection, the trusted module 100 preferably has an independent internal clock from which to measure time independent of the host computer. As described above, the external construction of the trusted module 100 is preferably either a PCMCIA-like device or a smart card-like device (not shown). The trusted module 100 is equipped to interface with a host computer using any type of bi-directional interface 130 such as for example a standard USB port or a 20-bit bus with three consecutive ports. Of course, the bit size of the bus is not critical to the present invention.

With further reference to Figure 3, the object machine (Level 1) executes the TMOS kernel of the trusted module, which controls the internal resources of trusted module and the logical security of trusted module (i.e., protection of secret internal objects from unauthorized reading or modification). In particular, the TMOS kernel supports the trusted module's user authentication processes. The TMOS kernel protects the internal integrity of the trusted module by encrypting the sensitive parameters of TMOS Kernel, which are preferably decrypted in RAM only at the time of use. Furthermore, a portion of the segments of the TMOS Kernel are distributed between ROM and NVRAM and are preferably concatenated only at the time of execution.

In the exemplary embodiment, the code of the TMOS Kernel is digitally signed. In an

Development of applications for use within the framework of the technology is described and shown in connection with Figures 8 and 9. In the exemplary embodiment of the protected application development, a set of program segments is identified. Of course, one skilled in the art will recognize that the selection of these segments is dependent on a particular application as a matter of design choice. This selection can also be performed automatically using some technological software. Moreover, although it is not necessary, it is preferable to protect segments without which the protected software would not perform its functions, or containing algorithms that are not known to the operators of the software. Furthermore, it is preferable that all functions related to the application of cryptographic algorithms and their parameters (keys) should also be protected. Along with the algorithm integrity, protected storage of keys is also a preferred feature.

Thus, the protected process is split into two new processes, interacting with each other – the “open” process, which is executed on the open architecture computer, and the “protected” process, which is executed using the trusted module.

With reference to Figure 8, an exemplary embodiment of a process for the preparation of a protected application for execution within the PEP-machine of the trusted module is shown. The source code 4 of a software product is divided into segments $S_1 \dots S_n$ from which segments desired to be protected 41 (e.g., those critical to program execution) are selected. Next, the segments to be executed using the trusted module are compiled using the CryptoCompiler 8. A private key 23 may be used as an additional parameter in compiling the selected segments. The CryptoCompiler is a program for translating the source code of the select segments of the protected program into “CryptoCode”. The CryptoCompiler uses a system of instructions that corresponds to the architecture and instruction set of a particular PEP-machine. The parameters of translation, including the set of instructions, their encoding and other elements of the architecture, can be established according to the additional

the RAM 11 of host computer 1 from a disk drive or other media 14 or received from a remote computer via a communication adapter 13. The software consists of segments S1, S2...Sn, a number of which ("open" segments, denoted as S1, S3, S5 in Fig. 6) are designed for execution using CPU 12 of the host computer, while the other part (protected segments, denoted as S2, S4, S6 in Fig. 1) are designed for execution using the PEP virtual machine 2. When executing the program the trusted module interacts with the host computer via interface 3.

The design of the Trusted Module 2 provides physical security sufficient to prevent external unauthorized access to the contents of the trusted module including the hardware and internal data areas. One of the possible implementations of the trusted module is a compact single-case device to be connected directly to the port connector of the host computer. Another implementation is a "smart card" form factor device with a set of standard interfaces, to be connected to a special adapter, which, in turn, is connected to the host computer.

The primary components of the Trusted Module are the CryptoInterpreter 21, a non-volatile memory 22 and a clock 24. The CryptoInterpreter 21 interprets and executes the commands of the code of the protected program being executed (i.e., the CryptoCode) received from the host computer. The Non-volatile memory 22 can store key(s) 23 used by the PEP Virtual Machine for deciphering of protected program code and can be used for protected storage of sensitive information between the working sessions against external reading/modification. Key(s) 23 should match the keys used when preparing the protected code segments and static program data (S2, S4, S6 in shown Figure 6).

Some of the functions of Clock 24 allow the trusted module:

(1) to the check time spent by the host computer while executing trusted module requests to read/write host computer RAM,

other media and loaded into the RAM of the host computer before execution of the program together with non-protected segments. They are fetched by the trusted module when necessary using the procedures which will be described further below.

On the logical level, an exemplary process of program execution is illustrated in Figure 10. While executing the program, at least two processes are generated: the "open" process **A**, executed by the host computer CPU and the protected process **B**, executed on the PEP Virtual Machine. Because the protected segments are compiled using the unique instruction set of the CryptoCompiler, the executable code **B1** and state vector (data) **B2** of the protected process **B** are not available for reading and/or modification from the host computer. However, code **A1** and state vector **A2** of the open process **A** are available for reading and modification due to the open architecture of the host computer. While the program is executed, an interaction of open and protected processes using some inter-processor interaction mechanism (for example, using shared memory window or messages) can be performed, using a shared section of the state vector **C**. In this case, the protected process can send and receive input and output data, check the state vector of the open process, check the consistency of the state vector, and compare time parameters of processes using the independent clock of the trusted module. Mismatch of any of the controlled parameters of the process is detected and results in setting of the process protection violation flag **B21**. This event is also communicated to the open process where it results in setting of the process protection violation flag **A21**. Possible reaction to the detection of security violation can be in particular erasing of the key information, rendering the PEP Virtual Machine unusable, erasing of sensitive information, blocking of processes or other actions.

Interaction between the host computer and the trusted module when executing a program developed using the PEP technology of the exemplary embodiment is illustrated by Figure 11. When executing the protected process, the information exchange between the

(1) Computation of Message Authentication Codes (MACs), using secret keys stored inside trusted module; and/or

(2) Computation of digital signatures using PEP virtual machine public/private key pair.

To transfer input data to the protected process and receive the results back to the "open" process of the host computer, information flow G is used, which is not enciphered. The read/write access to internal NVRAM 22 is given only to the protected process using special instructions. To increase security, data can be stored in NVRAM in enciphered form.

Thus, the described structure of interaction of the trusted module and the host computer can serve as a base for at least the following:

- protection of code and data of the process being executed from unauthorized reading by enciphering them when stored in the host computer memory;

- detection of unauthorized modification of code and process data when stored in host computer memory using MACs or digital signatures;

- independent (from the host computer clock) checking of time parameters of the protected process;

- inaccessible (for unauthorized modification) storage of critical data in NVRAM of the trusted module with possible transfer of the data to another computer together with the trusted module.

When the trusted module detects an attempt to influence the course of the process (e.g., detects attempts to change code or data while using MACs or detects discrepancy in time parameters of the executed process compared to the expected ones or detects other indications of unauthorized interference in the process execution) the control program of the trusted module can disable the PEP Virtual Machine by erasing the code de/enciphering key or by setting an event flag of a detected attempt of external interference in trusted module

machine, so that the same word located at different addresses is represented by different words after enciphering.

It should also be noted that a fixed key is necessary for the enciphering of executable code of the program and constants. This key, therefore, must be defined at the preparation (compiling) stage of the protected program. To encode the working data, generated and used only by the protected process at the time of execution, it is possible to use a new key for each new working session. For instance, a new enciphering key for the working data of the virtual machine may be designed at every new run of the program (when initializing the virtual machine). This key can be generated for example by using a built-in random number generator.

An additional measure aimed at hampering of the analysis of the algorithm and the state of the executed process when using word-by-word retrieval can be scrambling of words addresses when storing them in the memory of host computer. In this case, the words positioned in address space of the virtual machine are mapped to pseudorandomly located words in the address space of the host computer allocated for their storage. When executing a linear segment of the protected program, the requests of the virtual machine to the memory of the host computer appear to be random to the external observer. The above-mentioned reasoning related to the enciphering of working data of protected process also applies to scrambling of addresses of the working data generated and used only by the protected process at time of execution, i.e., it is advantageous to use a newly generated pseudorandom key for address scrambling of data in each new working session.

To provide data exchange with the "open" process of the host computer (communication of input data and results of calculations) special requests supported by a driver and the TMOS are used, or an open memory window is allocated in the logical address space of the virtual machine and no enciphering and address scrambling is used for reading

An additional measure aimed at hampering program logic analysis can be the generation of requests, which are independent of the work of the protected process, to the host computer memory (to pseudorandom addresses at pseudorandom time intervals for reading/writing to an unused memory region) from the trusted module. By way of example only, addresses above some address in the address space of the Virtual Machine may be used. These operations result in some additional load on the interface between the host computer and the trusted module and, therefore, they should not be performed too often.

An additional measure aimed at hampering of program logic analysis can be the redundant recording of a number of copies of executable code and static data of the program into a window allocated in the main memory of host computer, so that at a specific address in the address space of the Virtual machine there would be the first copy, then - the second copy, etc. until all the allocated address space is consumed. Taking into account address scrambling, the copies will be "randomly" mixed in the memory window of the host computer allocated for trusted module programs and data. Considering the dependency of the enciphering algorithms on the address in address space of the virtual machine, the words of different copies will be mapped to different words after enciphering. When executing the program, the control program of the trusted module can select words of arbitrary copies of program using a pseudorandom algorithm for copy selection. A simple design can be the separation of copies in logical address space of the Virtual Machine by a fixed distance dA and virtual machine's reading, instead of the word at the address a , the word at the address $a + dA * r$, where r is a number generated by random number generator and r is less than the number of copies of code and static data of the program stored in the address space of the virtual machine.

An additional measure aimed at hampering of program logic analysis can be checking of the host computer reaction times to requests of word read/write by the OS of the trusted

trusted module is not so important as it is when adding a MAC to each word in the word-by-word exchange scheme.

When using code fragments of small size, it is possible to implement the above indicated schemes of address scrambling, the only difference being that the whole fragments are rearranged instead of words.

To increase cryptographic security it is possible to introduce an additional parameter -- address of the protected fragment -- to ciphering algorithms.

It should be understood that in addition to the variants listed above it is also possible to implement intermediate variants, such as word-by-word access with cache buffer in the trusted module RAM to optimize data exchange with the host computer or organizing a paged memory with swapping from/to the host computer RAM.

Protection from unauthorized reading of the executable code (algorithm) and data

The protected software, prepared using the PEP-technology, includes protected fragments or segments, as described above. These fragments are preferably enciphered using symmetric algorithm using a secret key. While executing the program, the protected fragments of code and data are stored in the host computer RAM only in an enciphered form. Deciphering is performed by the trusted module in the course of execution (interpretation) of executable code or data retrieval. Deciphering keys are stored in NVRAM or ROM of the trusted module and are not available for external reading and modification due to the physically and logically protected design of the trusted module. This makes the reading of the program code before, after and during its execution impossible.

Protection from unauthorized alternation of executable code or static data of the program (authentication of program and data)

When using MACs to check the integrity of protected code and data, the code fragment or individual words (depending on the used scheme of protected execution) are

particular, program counter (address of the next instruction to be executed);

- contents of NVRAM **B22** (critical information of long-term storage);
- the working data of the process **5**, stored in host computer RAM **11**;
- data **L** received or transmitted to the “open” process of the host computer.

Data **B21** and **B22** are stored in the physically and logically protected trusted module **2** and, therefore, are not available for external observation. The program counter of the virtual machine (address of the next instruction to be executed) can be indirectly located using the analysis of the sequence of requests to the host computer memory. But the use of the above described address scrambling, variable address mappings and additional pseudorandom requests to the host computer memory from the trusted module will complicate the localization of the address.

Data **I** is stored not within the physically secure trusted module **2**, but rather in the RAM of the host computer **11**, and is potentially available for observation. Nevertheless, since it is stored in an enciphered form using a secret enciphering key (which can be either fixed or unique for each new working session) a “sensible” reading of this data from the host computer is impossible.

Data **L** is designed for information exchange with the open process of host computer and, therefore, cannot be enciphered. However when the functional interface between the “open” and protected parts of the program is organized correctly pursuant to the teachings herein and application specific design needs, the security of the system should not be compromised.

Protection from the introduction of unauthorized changes into the current state of the process being executed at some point of its execution

Referring to Figure 14, we should note that the internal registers of the virtual machine **B21** preferably include, in particular, the program counter of the virtual machine (address of

of work performed by the host computer program during a particular time period. On the other hand, the use of some means (e.g., debugger) to study the logic of the protected program by a hacker would have a high probability of increasing request's execution time. In some applications, the time of execution of a particular fragment of protected program can have an important value itself and it is necessary to prevent artificial expansion of this time. When loading large fragments for execution into the trusted module, the invariability of time parameters during execution of the fragment is ensured by the physical protection of the trusted module and its independence from the clock rate of the host computer. In instances where word-by-word retrieval is used, the invariability can be ensured to some degree by controlling the time of fulfillment of requests for reading/writing of words to and from the host computer memory.

Second, during execution of an application, an interaction of the protected process and the open process is performed. Thus, the time of execution of particular open fragments of the program should also be checked. This function is performed by the protected process, which, if necessary, can check (using the Trusted Module's clock) time intervals between specific events of the open process (for example, between calls to functions of the protected program), as well as match the internal time of the Trusted Module with the system time of the host computer to check the synchronism (keeping time scale and uniformity) of processes' execution.

Finally, the trusted module clock can be used to calculate the time of usage of protected software, which (combined with NVRAM) can be used for software rental and other purposes.

Monitoring and checking of the open fragments of the process being protected

In order to ensure the integrity of the whole process, it is necessary to monitor fragments or segments of the process being executed on the host computer (the "open"

Protection of the non-volatile memory from unauthorized access

The read/write operations data from and to NVRAM can be performed only by the protected process. These operations are commands of the PEP virtual machine of the trusted module. Accordingly, when using MACs to provide authentication of executable code and data of protected program, it is possible with high probability to avoid the possibility of substitution of code or data fragments, resulting in reading/writing of NVRAM.

Provision to make each copy of software unique with ability to provide means of access control.

When using unique keys for each of the produced copies of a program (for enciphering, address scrambling and/or generation of MACs of executable code and static data of protected programs), the copy can be executed only using the trusted module with the corresponding keys.

The programmable structure of the crypto-compilation process and possible control over the keys distribution potentially allows for more flexible control over protected programs preparation, distribution, and execution. For instance, it is possible to create a set of programs, which can be executed only using the specific trusted module, or to create a single program, which can be executed using a specified set of trusted modules.

Along with binding of a specific copy of software to a specific device at the time of software compilation, it is also possible to bind software, manufactured separately and delivered to the user later, using communication networks or other transport. If this ability is supported, the new software is deciphered, its integrity is checked using corresponding public key of software provider, and the protected software is stored, ready to execute, in the host computer, in the re-encrypted form. During this re-encryption process, deciphered code or data do not appear outside the physically protected trusted module.

Using the physically protected security kernel of trusted module and/or dynamically

WE CLAIM:

1. A method of securely executing a software application on a host computer and coprocessor, the method comprising:

selecting a first set of segments of the software application to be executed on an open architecture system;

selecting a second set of segments of the software application to be executed only by a closed architecture system;

compiling the first set of segments using a first compiler into a first set of code;

compiling the second set of segments using a second compiler into a second set of code;

linking the first and second sets of code;

executing the first set of code on the open architecture system of the host computer; and

executing the second set of code only on the closed architecture system of the coprocessor.

2. The method of claim 1, wherein the step of processing the software application further comprises processing results of the execution of the first and second sets of code through interaction between the host computer and the coprocessor.

3. The method of claim 1, wherein the step of processing the software application further comprises:

storing at least the second set of code in a memory of the host computer;

transmitting a portion of the second set of code to the coprocessor; and

interpreting the portion of the second set of code in the coprocessor.

second set of code in a segment-by-segment mode.

12. The method of claim 11, wherein the transmitting of the portion of the second set of code in segments further comprises:

transmitting an authorization code along with each of the segments; and

checking the authorization code upon receipt of one of the segments in the coprocessor.

13. The method of claim 1, further comprising enciphering the second set of code using a key corresponding to a key stored in a memory of the coprocessor.

14. The method of claim 1, further comprising storing the linked first and second sets of code on a machine readable storage device.

15. The method of claim 1, further comprising making the linked first and second sets of code accessible from a remote location.

16. The method of claim 1, further comprising checking a time period during processing between issuance of a request by the coprocessor and fulfillment of the request by the host computer.

17. The method of claim 16, wherein the time period is measure by a clock integrated within the coprocessor.

18. The method of claim 1, further comprising matching a time value of a clock of the coprocessor with a time value of a clock of the host computer during processing.

19. The method of claim 1, further comprising calculating a time of usage of the software application using a clock of the coprocessor.

20. A system for securely executing a protected application, a first portion of the protected application compiled using an open architecture instruction set and a second portion of the protected application compiled using a protected instruction set, the system comprising:

28. The system of claim 26, wherein each word transmitted to the trusted module is in an enciphered form.

29. The system of claim 26, wherein each word is transmitted to the trusted module along with an authorization code and the trusted module checks the authorization code.

30. The system of claim 20, wherein the trusted module further comprises:
a communication interface for communicating with the host computer;
a memory for storing keys for deciphering the second portion of the protected application; and
a clock.

31. The system of claim 30, wherein the communication interface is a universal serial bus.

32. The system of claim 30, wherein the communication interface is a parallel port.

33. The system of claim 32, wherein the parallel port is on the IEEE-1284 type.

34. The system of claim 30, wherein the memory is a non-volatile memory.

35. The system of claim 30, wherein the keys stored in the memory match keys used to encipher the second portion of the protected application.

36. The system of claim 30, wherein the clock checks a time interval spent by the host computer to execute requests of the trusted module.

37. The system of claim 30, wherein the coprocessor limits usage of the protected application based on a time value measured by the clock.

38. The system of claim 37, wherein the time value is predefined.

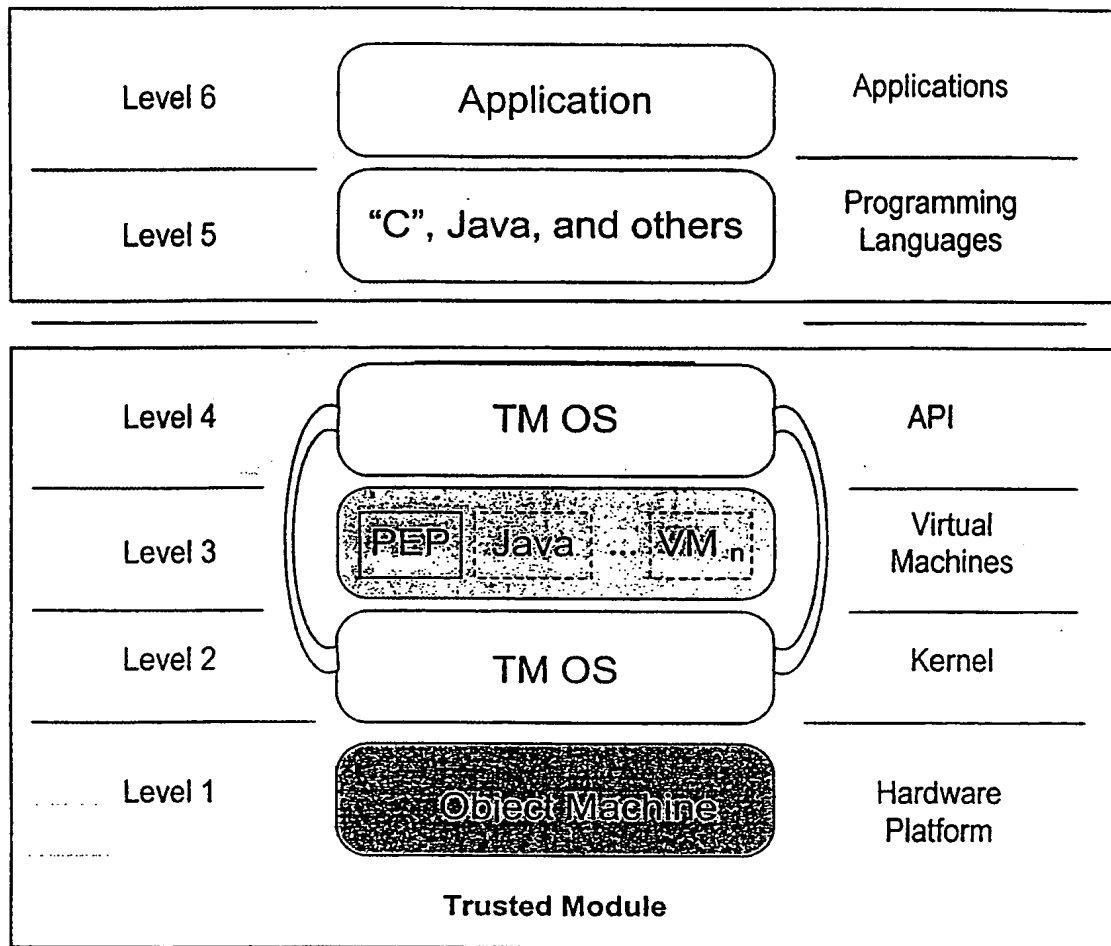
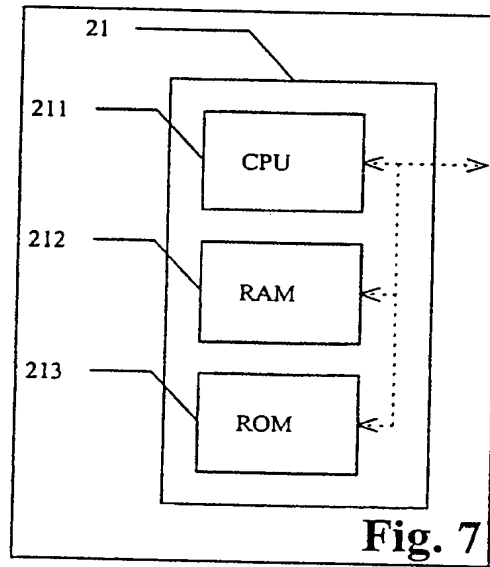
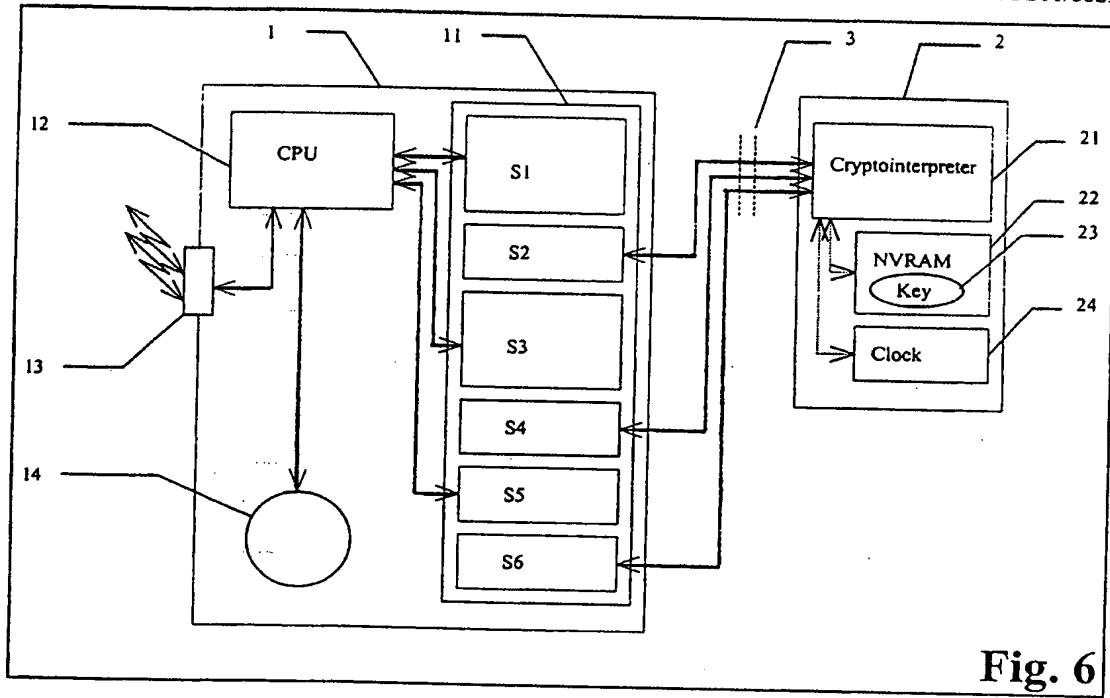
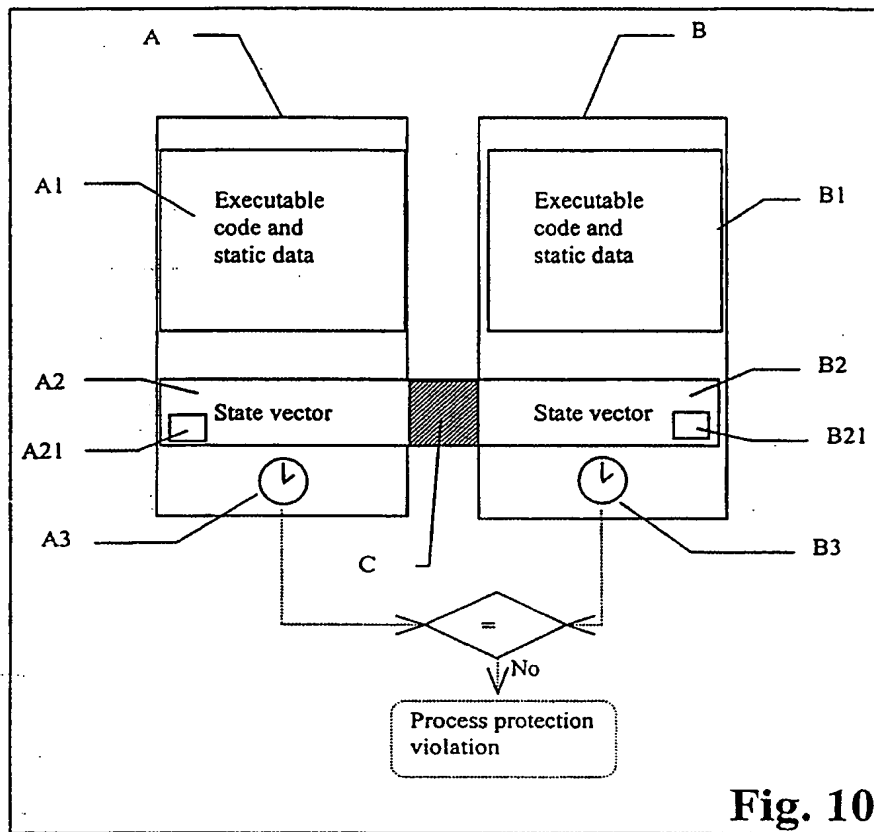
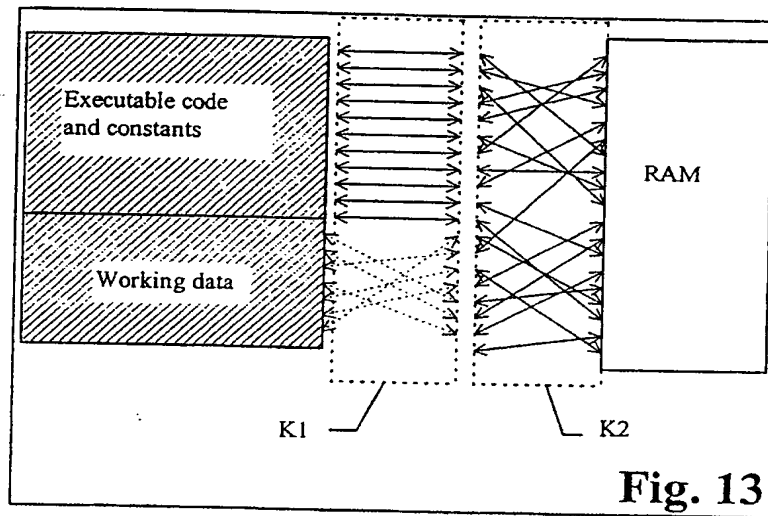
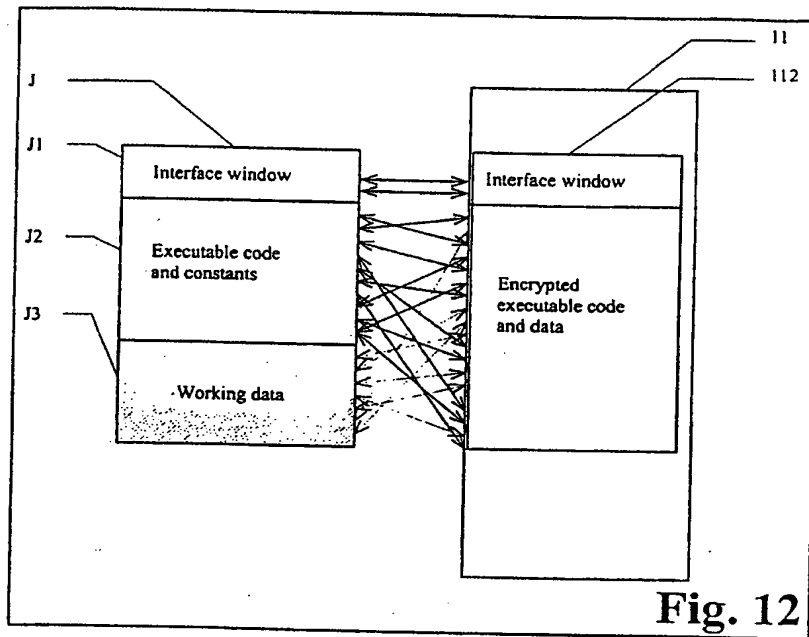


Fig. 3







INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/06599

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 9/45, 11/30, 12/14; H04L 9/00, 9/32; H04K 1/00

US CL : 713/189, 190, 200, 201; 725/31; 380/251; 717/1, 4, 5

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 713/189, 190, 200, 201; 725/31; 380/251; 717/1, 4, 5

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US 6,012,144 A (PICKETT) 04 JANUARY 2000, COL. 1, LINE 48 THROUGH COL. 2, LINE 2, COL. 2, LINES 48-65, COL. 4, LINES 15-37, 56-64, AND COL. 7, LINES 1-12	1-8, 10-38 — 9
X, P	US 6,125,186 A (SAITO ET AL) 26 SEPTEMBER 2000, COL. 11, LINES 46-51, COL. 4, LINES 50-53, COL. 12, LINES 4-9	1-7, 10-17, 20-38
Y	US 5,666,516 A (COMBS) 09 SEPTEMBER 1997, COL. 8, LINE 65 THROUGH COL. 9, LINE 10	9

☐ Further documents are listed in the continuation of Box C.☐ See patent family annex.

* Special categories of cited documents:	*T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
A document defining the general state of the art which is not considered to be of particular relevance	*X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
E earlier document published on or after the international filing date	*Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
L document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	*A* document member of the same patent family
O document referring to an oral disclosure, use, exhibition or other means	
P document published prior to the international filing date but later than the priority date claimed	

Date of the actual completion of the international search

04 MAY 2001

Date of mailing of the international search report

01 JUN 2001

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231

Facsimile No. (703) 305-3230

Authorized officer

CHRISTOPHER A REVAK

Telephone No. (703) 305-9618